

# From C to Rust: Bringing Rust Abstractions to Embedded Linux

Fabien Parent, Linaro

Embedded Open Source Summit 2024, Seattle, WA



# Introduction

- Senior Embedded Engineer - Qualcomm Engineering Team of Linaro
  - Working from British Columbia, Canada
  - Helping companies building products on top of Qualcomm SoCs
- More than 10 years experience in Embedded Linux
- Contributor to:
  - Linux Kernel
  - U-Boot
- [linkedin.com/in/fabienparent](https://www.linkedin.com/in/fabienparent)

# Linaro is the software engine of the Arm Ecosystem

Linaro empowers rapid product deployment within the dynamic Arm Ecosystem.

- Our cutting-edge solutions, services and collaborative platforms facilitate the swift development, testing, and delivery of Arm-based innovations, enabling businesses to stay ahead in today's competitive technology landscape.
- Our expertise and contributions spread from Testing & LTS, Security, Cloud & Edge Computing, IoT, AI, CI/CD, Toolchain and Virtualization to vertical projects like Windows on Arm and Android Ecosystem enabling and maintenance.
- Linaro fosters an environment of collaboration, standardization and optimization among businesses and open source ecosystems to accelerate the deployment of Arm-based products and technologies along with representing a pivotal role in open source discovery and adoption.

**Linaro has enabled trust, quality and collaboration since 2010**

# Why having Rust in the kernel?

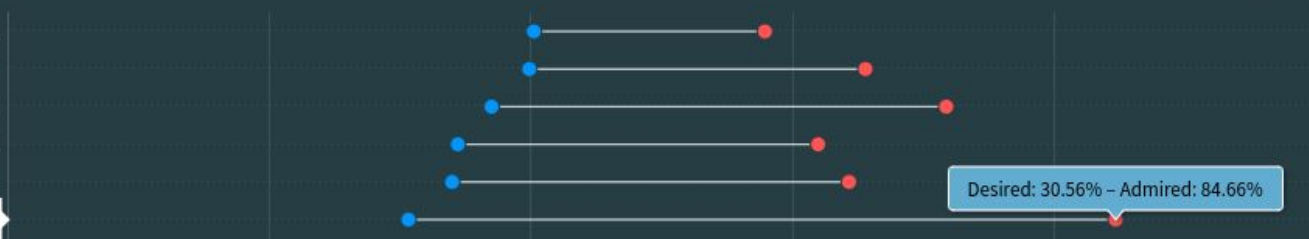
# Probably not for that reason...

## Programming, scripting, and markup languages

**Rust** is the most admired language, more than 80% of developers that use it want to use it again next year. Compare this to the least admired language: MATLAB. Less than 20% of developers who used this language want to use it again next year.

87,510 responses

JavaScript  
Python  
TypeScript  
HTML/CSS  
SQL  
Rust



Source: <https://survey.stackoverflow.co/2023/>

# Why adding Rust to the kernel?

## C Programming Language

- ~50 years old and hasn't changed much since C89
- Simple syntax, easy to learn
- Efficient
- Low-level control
- Easy to write code full of UB
  - Lack of memory management
- ... or full of unwanted behavior
  - Implicit type conversion

## Rust Programming Language

- ~10 years old, new release every 6 weeks
- Deep learning curve:
  - Ownership & Borrowing
- Efficient
- Force you to handle errors
- Efficient
  - Compile time checks
  - 0-cost abstractions

# Why adding Rust to the kernel?

One of the strongest argument of using Rust is its safety

- Compiler check for unsafe memory access, concurrency issue, ...

```
error[E0133]: dereference of raw pointer is unsafe and requires unsafe function or block
--> src/main.rs:4:5
4 |     *ptr = 5;
  |     ^^^^^ dereference of raw pointer
= note: raw pointers may be null, dangling or unaligned; they can violate aliasing rules and cause data races: all of these are undefined behavior
```

Must use the “unsafe” keyword when writing code that Rust compiler is unable to validate or that use pointers

```
unsafe impl Send for MySafeData {}
unsafe impl Sync for MySafeData {}
```

```
let ptr: *mut u32 = std::ptr::null_mut();

unsafe {
    *ptr = 5;
};
```

# Rust usage in kernel

- Only for writing device drivers
  - Clang/LLVM does not support as many architectures as GCC, so core frameworks cannot depend on Rust
  - GCC Rust Frontend is in the work but not yet ready
- Support is still considered as experimental



# Drawback from current Rust usage

- Major vulnerabilities are not in device drivers but in the subsystems, such as Networking, Filesystems, USB, and any other that process buffers
- Adds more complexity to maintenance, if we don't want things to break, now Maintainers needs to know two languages
- Abstractions becomes a new source of possible bugs and potentially vulnerabilities, which would defeat the argument of using Rust for added security



You

Give me a list of the last 5 vulnerabilities in the Linux kernel you are aware of



ChatGPT

As of my last update in January 2022, I can't provide real-time information on the latest vulnerabilities. However, I can give you examples of types of vulnerabilities that have been found in the past:

1.

**CVE-2021-3493:** A flaw in the Linux kernel's filesystem layer that allowed a local attacker to gain elevated privileges.

2.

**CVE-2021-34556:** A vulnerability in the Linux kernel's networking subsystem that could lead to denial-of-service (DoS) attacks or potentially arbitrary code execution.

3.

**CVE-2021-35265:** A flaw in the Linux kernel's handling of certain types of network packets that could lead to a denial-of-service condition.

4.

**CVE-2021-33200:** A vulnerability in the Linux kernel's USB subsystem that could allow a physically proximate attacker to execute arbitrary code or cause a denial of service.

5.

**CVE-2021-31829:** A security issue in the Linux kernel's HID subsystem, which could allow an attacker with physical access to a system to escalate privileges.

Please note that these are examples from the past and may not reflect the latest vulnerabilities. For up-to-date information, I recommend referring to security advisories from Linux distribution vendors or other trusted sources.

# My Rust's journey

# Disclaimer

- I'm **\*NOT\*** a Rust expert
- This presentation is narrating my journey learning Rust to contribute to that part of the kernel

# How this story started?



[parent](#) | [Log out](#) | [\(Linaro\)](#)

## A GPIO driver in Rust

[Posted July 19, 2021 by [coln](#)]

As an example of what a "real" device driver in Rust would look like, Wedson Almeida Filho has [posted a translation](#) of the PL061 GPIO driver alongside the original. For ease of reading, the resulting HTML has been reformatted a bit and placed below; viewing in a wide window is recommended.

**Content**  
[Weekly Edition](#)  
[Archives](#)  
[Search](#)  
[Kernel](#)  
[Security](#)  
[Events calendar](#)  
[Unread comments](#)  
  
[LWN FAQ](#)  
[Write for us](#)  
  
**Editorial**  
[Return to the Briefs page](#)

### C version

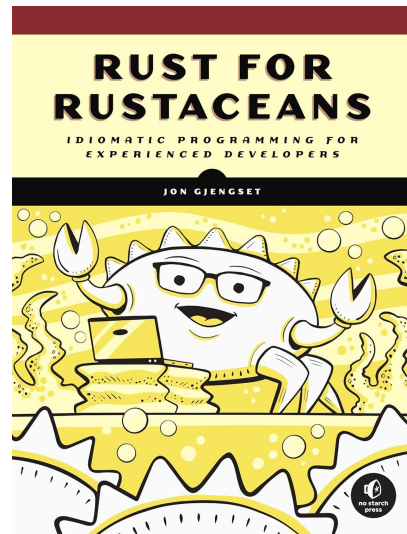
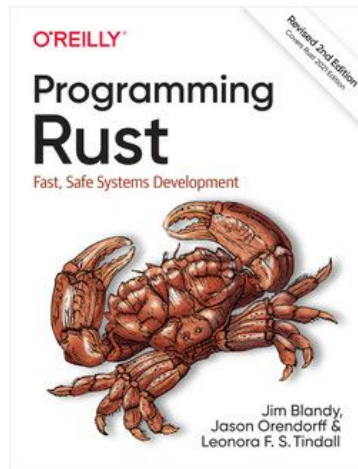
```
1 // SPDX-License-Identifier: GPL-2.0-only
2 /*
3  * Copyright (C) 2008, 2009 Provigent Ltd.
4  *
5  * Author: Baruch Siach <baruch@tkos.co.il>
6  *
7  * Driver for the ARM PrimeCell(tm) General Purpose Input/Output (PL061)
8  *
9  * Data sheet: ARM DDI 0190B, September 2000
10 */
11 #include <linux/spinlock.h>
12 #include <linux/errno.h>
13 #include <linux/init.h>
14 #include <linux/io.h>
15 #include <linux/ioport.h>
16 #include <linux/interrupt.h>
17 #include <linux/irq.h>
18 #include <linux/irqchip/chained_irq.h>
19 #include <linux/module.h>
20 #include <linux/bitops.h>
21 #include <linux/gpio/driver.h>
22 #include <linux/device.h>
23 #include <linux/amba/bus.h>
24 #include <linux/slab.h>
25 #include <linux/pinctrl/consumer.h>
26 #include <linux/pm.h>
27
28 #define GPIODIR 0x400
29 #define GPIODIS 0x404
30 #define GPIODISE 0x408
31 #define GPIOIEV 0x40C
32 #define GPIOIE 0x410
33 #define GPIODIS1 0x414
34 #define GPIODIS2 0x418
35 #define GPIOIC 0x41C
36
37 #define PL061_GPIO_NR 8
38
39 #ifdef CONFIG_PM
40 struct pl061_context_save_regs {
41     u8 gpio_data;
42     u8 gpio_dir;
43     u8 gpio_ie;
44     u8 gpio_ibe;
45     u8 gpio_iev;
46     u8 gpio_ie;
47 };
48 #endif
49
50 struct pl061 {
51     raw_spinlock_t lock;
52     void __iomem *base;
53     struct gpio_chip gc;
54     struct irq_chip irq_chip;
55     int parent_irq;
56
57 #ifdef CONFIG_PM
58     struct pl061_context_save_regs csave_regs;
59 #endif
60 };
61
62
```

### Rust version

```
1 // SPDX-License-Identifier: GPL-2.0
2
3 /// Driver for the ARM PrimeCell(tm) General Purpose Input/Output (PL061).
4 ///
5 /// Based on the C driver written by Baruch Siach <baruch@tkos.co.il>.
6
7 #![no_std]
8 #![feature(global_asm, allocator_api)]
9
10 use core::ops::DerefMut;
11 use kernel::{
12     amba, bit, declare_id_table, device, gpio,
13     io_mem::IoMem,
14     irq::{self, IrqData, LockedIrqData},
15     power,
16     prelude::*,
17     sync::{IrqDisableSpinLock, Ref},
18 };
19
20 const GPIODIR: usize = 0x400;
21 const GPIODIS: usize = 0x404;
22 const GPIODISE: usize = 0x408;
23 const GPIOIEV: usize = 0x40C;
24 const GPIOIE: usize = 0x410;
25 const GPIODIS1: usize = 0x414;
26 const GPIOIC: usize = 0x41C;
27 const GPIO_SIZE: usize = 0x1000;
28
29 const PL061_GPIO_NR: u16 = 8;
30
31 #![derive(Default)]
32 struct ContextSaveRegs {
33     gpio_data: u8,
34     gpio_dir: u8,
35     gpio_ie: u8,
36     gpio_ibe: u8,
37     gpio_iev: u8,
38     gpio_ie: u8,
39 }
40
41 #![derive(Default)]
42 struct PL061Data {
43     csave_regs: ContextSaveRegs,
44 }
45
46 struct PL061Resources {
47     base: IoMem<GPIO_SIZE>,
48     parent_irq: u32,
49 }
50
51 struct PL061Registrations {
52     gpio_chip: gpio::ChipRegistration<PL061Device>,
53 }
54
55 type DeviceData = device::Data<PL061Registrations, PL061Resources, IrqDisableSpinLock<PL061Data>>;
56
57 struct PL061Device;
58
59 impl gpio::Chip for PL061Device {
60     type IrqChip = Self;
```

# Learning Rust

- <https://www.rust-lang.org/learn>
  - The Book
  - Rust by Example
  - APIs documentation
- Programming Rust, Blandy, Orendorff, Tindall, O'REILLY
- Rust for Rustaceans: Idiomatic Programming for Experienced Developers, by Gjengset, No Starch Press



# Practicing

1. Tried to write diffectly Rust code in the kernel since it was the goal
  - Difficult because there was not that much existing code to learn from
  - was not yet writing idiomatic Rust code
2. Rewrote *\*all\** my custom Python, Bash, C tools in Rust as practice
3. Back to the Linux

# State of Rust for Linux at the time

- Rust initial patchset was merged into v6.1
  - Can write and compile Rust code in the kernel
  - Can generate Rust docs from the kernel
  - Can run KUnit for Rust tests
  - Can compile some Rust code as a kernel module
- No other abstractions were merged upstream
- Some abstractions are in a downstream unmaintained branch:  
<https://github.com/Rust-for-Linux/linux/tree/rust>

# State of Rust for Linux at the time?

```
// SPDX-License-Identifier: GPL-2.0

//! Rust minimal sample.

use kernel::prelude::*;

module! {
    type: RustMinimal,
    name: "rust_minimal",
    author: "Rust for Linux Contributors",
    description: "Rust minimal sample",
    license: "GPL",
}

struct RustMinimal {
    numbers: Vec<i32>,
}

impl kernel::Module for RustMinimal {
    fn init(_name: &'static CStr, _module: &'static ThisModule) -> Result<Self> {
        pr_info!("Rust minimal sample (init)\n");
        pr_info!("Am I built-in? {}\n", !cfg!(MODULE));

        let mut numbers = Vec::new();
        numbers.try_push(72)?;
        numbers.try_push(108)?;
        numbers.try_push(200)?;

        Ok(RustMinimal { numbers })
    }
}

impl Drop for RustMinimal {
    fn drop(&mut self) {
        pr_info!("My numbers are {:?}\n", self.numbers);
        pr_info!("Rust minimal sample (exit)\n");
    }
}
```



# A lot more is happening, but ...

<https://github.com/tgross35/RFL-patch-registry>

RFL-patch-registry / README.md

Preview Code Blame 162 Lines (543 loc) · 12.9 KB

<https://rust-for-linux.zulipchat.com/>

Note that things change frequently! This is a best effort list, PRs or issues are welcome if you notice something out of sync.

Target Area	Status	Author	
AArch64 support	merged	Jamie C.	
Loongarch support	merged	Rui W.	
RISC-V support	review	Conor D.	
x86_64 support	merged		
condvar	merged		<a href="https://elixir.bootlin.com/linux/v6.4/sc">https://elixir.bootlin.com/linux/v6.4/sc</a>
cred	review	Alice R.	
crypto	RFC	Tomo F.	
debugfs	testing	Adam BK.	<a href="https://rust-for-linux/linux#885">Rust-for-Linux/linux#885</a>
debugfs	experimental	Fabien P.	<a href="https://rust-for-linux/linux#1041">Rust-for-Linux/linux#1041</a>
delay	testing	Tomo F.	<a href="https://rust-for-linux/linux#920">Rust-for-Linux/linux#920</a>
device	experimental	Gary G.	<a href="https://github.com/nbddd0121/linux/tr">https://github.com/nbddd0121/linux/tr</a>
debugfs	experimental	Fabien P.	<a href="https://github.com/Fabio/linux/tree/fp">https://github.com/Fabio/linux/tree/fp</a>
driver	experimental	Martin RR., Wedson AF.	<a href="https://github.com/YakoYakoYoko">https://github.com/YakoYakoYoko</a>
dma	testing	Tomo F.	<a href="https://rust-for-linux/linux#901">Rust-for-Linux/linux#901</a>
drm	RFC	Lina	
drm-asahi (apple Si)	RFC	Lina, Alyssa R.	<a href="https://github.com/AsahiLinux/linux/t">https://github.com/AsahiLinux/linux/t</a>
drm-nova (nouveau rewrite)	experimental	Daniilo K., Dave A.	<a href="https://gitlab.freedesktop.org/dakr/nv">https://gitlab.freedesktop.org/dakr/nv</a>
drm-panel, drm-dsi	experimental	Fabien P.	<a href="https://github.com/Fabio/linux/tree/fp">https://github.com/Fabio/linux/tree/fp</a>
ethernet	testing	Amélie	<a href="https://rust-for-linux/linux#1014">Rust-for-Linux/linux#1014</a>
file, misc	RFC	Alice R., Wedson AF.	

RFL-patch-registry / README.md

Preview Code Blame 162 Lines (543 loc) · 12.9 KB

Target Area	Status	Author	
file, misc	review	Alice R., Wedson AF.	
fnnode	testing	Vinay V.	<a href="https://rust-for-linux/linux#925">Rust-for-Linux/linux#925</a>
gpiod/consumer	experimental	Fabien P.	<a href="https://github.com/Fabio/linux/commi">https://github.com/Fabio/linux/commi</a>
gfp/highmem/page	review	Alice R.	
i2c	testing	Finn B.	<a href="https://rust-for-linux/linux#946">Rust-for-Linux/linux#946</a>
i2c	experimental	Fabien P.	<a href="https://github.com/Fabio/linux/tree/fp">https://github.com/Fabio/linux/tree/fp</a>
io_pgtable	testing	Lina	<a href="https://github.com/Rust-for-Linux/linux/pull/952/commits/f476b2">https://github.com/Rust-for-Linux/linux/pull/952/commits/f476b2</a>
io_resource	testing	Maciej F.	<a href="https://rust-for-linux/linux#682">Rust-for-Linux/linux#682</a>
ioctrl	merged		<a href="https://git.kernel.org/pub/scm/linux/k">https://git.kernel.org/pub/scm/linux/k</a>
jiffies	testing	Maria C.	<a href="https://rust-for-linux/linux#982">Rust-for-Linux/linux#982</a>
kuid	review	Alice R.	
mm		Alice R., Wedson AF.	<a href="https://github.com/Darksonn/linux/cc">https://github.com/Darksonn/linux/cc</a>
napi	testing	Amélie	<a href="https://rust-for-linux/linux#1018">Rust-for-Linux/linux#1018</a>
net_device	RFC	Tomo F.	
null block	RFC	Andreas H.	
of	experimental	AsahiLinux	<a href="https://github.com/AsahiLinux/linux/t">https://github.com/AsahiLinux/linux/t</a>
pci	testing	Andreas H.	<a href="https://github.com/Rust-for-Linux/lini">https://github.com/Rust-for-Linux/lini</a>
phy	merged	Tomo F.	<a href="https://git.kernel.org/pub/scm/linux/k">https://git.kernel.org/pub/scm/linux/k</a>
platform	testing	Rust GH	<a href="https://github.com/Rust-for-Linux/lini">https://github.com/Rust-for-Linux/lini</a>
pm_runtime	testing	Maciej F.	<a href="https://rust-for-linux/linux#700">Rust-for-Linux/linux#700</a>
poll	review	Alice R.	
ramfs	abandoned	Fox C.	<a href="https://rust-for-linux/linux#409">Rust-for-Linux/linux#409</a>
rbtree	review	Alice R.	

RFL-patch-registry / README.md

Preview Code Blame 162 Lines (543 loc) · 12.9 KB

Target Area	Status	Author	
regmap	experimental	Fabien P.	<a href="https://github.com/Fabio/linux/tree/fp">https://github.com/Fabio/linux/tree/fp</a>
regulator/consumer	experimental	Fabien P.	<a href="https://rust-for-linux/linux#1040">Rust-for-Linux/linux#1040</a>
regulator/driver	experimental	Fabien P.	<a href="https://github.com/Fabio/linux/commi">https://github.com/Fabio/linux/commi</a>
reset	testing	Nikos M.	<a href="https://rust-for-linux/linux#933">Rust-for-Linux/linux#933</a>
rktit (apple)	testing	Lina	<a href="https://github.com/Rust-for-Linux/linux/pull/952/commits/f7708d">https://github.com/Rust-for-Linux/linux/pull/952/commits/f7708d</a>
scatterlist	RFC	Fox C.	
schedule	testing	Boqun F.	<a href="https://rust-for-linux/linux#861">Rust-for-Linux/linux#861</a>
secctx	review	Alice R.	
sk_buf	RFC	Tomo F.	
socket	RFC	Michele DR.	
spl	testing	Esteban B.	<a href="https://rust-for-linux/linux#264">Rust-for-Linux/linux#264</a>
sysfs	testing	Martin RR.	<a href="https://github.com/YakoYakoYoko">https://github.com/YakoYakoYoko</a>
tarfs	testing	Wedson AF.	<a href="https://rust-for-linux/linux#1037">Rust-for-Linux/linux#1037</a>
task	merged		<a href="https://elixir.bootlin.com/linux/v6.4/sc">https://elixir.bootlin.com/linux/v6.4/sc</a>
timer	testing	Maria C.	<a href="https://rust-for-linux/linux#982">Rust-for-Linux/linux#982</a>
thread	testing	Boqun F.	<a href="https://rust-for-linux/linux#109">Rust-for-Linux/linux#109</a>
usb	testing	Martin RR.	<a href="https://rust-for-linux/linux#884">Rust-for-Linux/linux#884</a>
user_ptr	review	Alice R.	
virtio	testing	Richard L.	<a href="https://rust-for-linux/linux#886">Rust-for-Linux/linux#886</a>
workqueue	merged	Alice R.	<a href="https://elixir.bootlin.com/linux/v6.7/sc">https://elixir.bootlin.com/linux/v6.7/sc</a>
v4l2	review	Daniel A.	
vfs	RFC	Wedson AF.	
vfs, ext2	experimental	Wedson AF.	<a href="https://github.com/wedsonaf/linux/cc">https://github.com/wedsonaf/linux/cc</a>
vfs, ext2	experimental	Tomo F.	<a href="https://github.com/fujita/linux/tree/fo">https://github.com/fujita/linux/tree/fo</a>

# A lot more is happening, but ...

- The basics for device driver development is not upstream, so most of these are not making any progress or looks abandoned
- Many other people seems interested in writing Rust device drivers but are discouraged because of the lack of abstraction upstream.
  - Requires them to not just write a driver but to write all the needed abstractions as well
- Some other people wants to contribute for the sake of contributing but don't have any real needs and end up rewriting existing driver in Rust which end up being NAKed.

# Some of the work being done

- DRM Abstractions
  - DRM driver for Apple M1 GPUs by Asahi Linux
  - DRM driver for Nvidia GPUs by Danilo Krummrich, Red Hat
- Android Binder rewrite by Alice Ryhl, Google
- Filesystems:
  - Puzzlefs by Ariel Miculas, Cisco
  - tarfs by Wedson Almedia Filho, Microsoft
- Networking

# Bringing Rust Abstractions to Embedded Linux

# Embedded Linux

- Most Rust work is done on x86-64 at the moment
- Key abstractions are not upstream or are missing
- Some abstractions are available but not based on master, and are part of old unmaintained branch.
  - Device driver abstraction
  - Platform driver abstraction
  - ...

# What do we need?

Has been upstreamed:

- ARM64 support

What's needed?

- |  |  |
|--|--|
| <ul style="list-style-type: none"><li>• Clock</li><li>• Pinctrl</li><li>• Runtime PM</li><li>• Regmap</li><li>• Regulator</li><li>• OF</li><li>• DMA</li></ul> | <ul style="list-style-type: none"><li>• IRQ</li><li>• GPIO</li><li>• Reset</li><li>• Rpmsg</li><li>• Busses such as I2C, PCIe, ...</li><li>• Platform driver</li></ul> |
|--|--|

# Constraints

- Maintainers don't want to merge abstractions without end-users
- Usually device drivers requires plenty of abstractions to be available
  - These abstractions are often written by different people, and are all in different states
  - Difficulties to get things reviewed for large patchset involving many subsystems abstractions

# Goal

- Write “simple” drivers which has a minimal dependency list to make it easier for upstreaming
- Use these drivers to help get “orphaned” abstractions merged
- Try to cover as many “core” APIs used by embedded devices through different drivers



# Regulator driver - ncv6336

- Link: <https://github.com/Fabo/linux/commits/fparent/rust-ncv6336/>
- Dependencies:
  - Platform driver (already present in old unmaintained `rust` branch)
  - Regulator driver
  - Regmap
  - I2C
  - OF

# DSI panel driver:

- Link: <https://github.com/Fabo/linux/commits/fparent/rust-truly-r63350>
- Dependencies:
  - Platform driver
  - Regulator consumer
  - DRM (Connector, Panel, MIPI DSI)
  - GPIO Consumer (only need a fraction of the C API)
  - OF

# Regmap

- Type-safe interface
- Re-using regmap\_field\_ API
- Macro-magic to generate most of the regmap callback based on register definitions

```
34 regmap::define_regmap_field_descs!(FIELD_DESCS, {
33   (pid, 0x3, READ, { value => raw([7:0], ro) }),
32   (rid, 0x4, READ, { value => raw([7:0], ro) }),
31   (fid, 0x5, READ, { value => raw([7:0], ro) }),
30   (progvsel1, 0x10, RW, {
29     voutvsel1 => raw([6:0], rw),
28     envsel1  => bit(7, rw),
27   }),
26   (progvsel0, 0x11, RW, {
25     voutvsel0 => raw([6:0], rw),
24     envsel0  => bit(7, rw),
23   }),
22   (pgood, 0x12, RW, { dischg => bit(4, rw) }),
21   (command, 0x14, RW, {
20     vselgt  => bit(0, rw),
19     pwmvsel1 => bit(6, rw),
18     pwmvsel0 => bit(7, rw),
17   }),
16   (limconf, 0x16, RW, {
15     rearm    => bit(0, rw),
14     rststatus => bit(1, rw),
13     tpwth    => enum([5:4], rw, {
12         Temp83C  = 0x0,
11         Temp94C  = 0x1,
10         Temp105C = 0x2,
9         Temp116C = 0x3,
8     }),
7     ipeak    => enum([7:6], rw, {
6         Peak3p5A = 0x0,
5         Peak4p0A = 0x1,
4         Peak4p5A = 0x2,
3         Peak5p0A = 0x3,
2     }),
1   })
70 });
```

# What's next?

- Upstreaming
- Next drivers
  - Cover other subsystems



# What did I learn?

# Abstractions are complex

```
+ pub fn new(  
+     dev: &Device<T::Driver>,  
+     type_ : u32,  
+     args: T::Args,  
+ ) -> Result<ARef<Self>> {  
+     let new: Pin<Box<Self>> = Box::try_pin_init(try_pin_init!(Self {  
+         connector: Opaque::new(bindings::drm_connector {  
+             helper_private: &Self::HELPER_FUNCS,  
+             ..Default::default()  
+         })),  
+         inner <- T::new(dev, args),  
+         _p: PhantomPinned  
+     ))?;  
+  
+     // SAFETY: FFI call with no special safety requirements  
+     to_result(unsafe {  
+         bindings::drm_connector_init(  
+             dev.drm.get(),  
+             new.raw_mut_ptr(),  
+             &Self::FUNCS,  
+             type_ as i32  
+         )  
+     })?;  
+  
+     // Convert the connector into an ARef so the caller has proper ownership over a refcount to  
+     // it. Also, the Box we consume here will be reconstructed in connector_destroy_callback()  
+     // once the connector's refcount drops to zero.  
+     // SAFETY: We currently hold ownership of the Box containing the connector and it's  
+     // refcount. As well, this operation will not move the contents of the Box.  
+     Ok(unsafe {  
+         ARef::from_raw(NonNull::new_unchecked(Box::into_raw(Pin::into_inner_unchecked(new))))  
+     })  
+ }
```

# Bugs will be lying in the abstractions

- Ownership of resources
- Linked Lists

# Linked List

```
4 struct mystruct {  
3     int a;  
2     int b;  
1 };  
5  
1 void mystruct_init(struct mystruct *s) {  
2     memset(s, 0, sizeof(*s));  
3 }
```

```
2 struct Registrations(Opaque<bindings::mystruct>);  
3  
4 impl kernel::Module for RustMinimal {  
5     fn init(_module: &'static ThisModule) -> Result<Self> {  
6  
7         let mystruct = Opaque::<bindings::mystruct>::uninit();  
8         unsafe {  
9             bindings::mystruct_init(mystruct.get());  
10        };  
11  
12        let registration = Registrations(mystruct);
```



# Linked List

```
8 struct mystruct {
7     int a;
6     int b;
+ 5     struct list_head list;
4 };
3
2 void mystruct_init(struct mystruct *s) {
1     memset(s, 0, sizeof(*s));
+ 9     INIT_LIST_HEAD(&s->list);
1 }
```

```
2 struct Registrations(Opaque<bindings::mystruct>);
3
4 impl kernel::Module for RustMinimal {
5     fn init(_module: &'static ThisModule) -> Result<Self> {
6
7         let mystruct = Opaque::<bindings::mystruct>::uninit();
8         unsafe {
9             bindings::mystruct_init(mystruct.get());
10        };
11
12        let registration = Registrations(mystruct);
13        unsafe {
14            let ptr = &(*registration.0.get()).list;
15            pr_info!("list = {:p}, list->next = {:p}, list->prev = {:p}",
16                    ptr, (*ptr).next, (*ptr).prev);
17        }
18    }
19 }
```

list = 0xffff80008001ba60  
list->next = 0xffff80008001ba48  
list->prev = 0xffff80008001ba48

# Summary

# Summary

- Error handling is so much better
  - Thanks for the *Result* type and *?* operator
  - Strongly typed: we don't need to guess how to interpret *int* or pointer return types
  - We must handle the returned values, no risk of “forgetting” to handle errors
- Source code for drivers is in general shorter and faster to write
- High confidence that the code is working if it compiles
- Need to be vigilant about Linked Lists
- Do not write feature-complete abstractions, only the pieces needed

# Want to contribute?

Links:

- <https://docs.kernel.org/rust/index.html>
- <https://rust-for-linux.com/>
- <https://rust-for-linux.github.io/docs/v6.8/kernel/>
- And join us on Zulip: <https://rust-for-linux.zulipchat.com/>

# Thank you

Visit [www.linaro.org](http://www.linaro.org)

Reach out to me at [fabien.parent@linaro.org](mailto:fabien.parent@linaro.org)





# Q & A